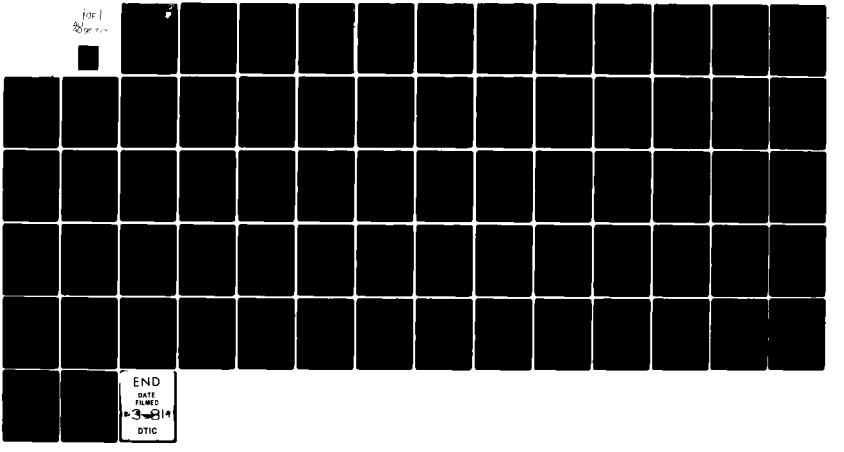
AD-A095 717 PATTERN ANALYSIS AND RECOGNITION CORP ROME NY F/G 9/2
ADVANCED QUERY FACILITY.(U)
NOV 80 C P MAH F30602-79-C-0174
UNCLASSIFIED PAR-80-53 RADC-TR-80-356 NL



LEVEL II

AD A096717

RADC-TR-80-356
Final Technical Report
November 1980

ADVANCED QUERY FACILITY

Pattern Analysis and Recognition Corporation

Dr. Clinton P. Mah

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
MAR 3 1981
S D
D

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-356 has been reviewed and is approved for publication.

APPROVED:

ZBIGNIEW L. PANKOWICZ
Project Engineer

APPROVED:

OWEN R. LAWTER, Colonel, USAF
Chief, Intelligence & Reconnaissance Division

FOR THE COMMANDER:

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDT) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
(18) 1. REPORT NUMBER: RADC-TR-80-356	2. GOVT ACCESSION NO. AD-A095 717	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): ADVANCED QUERY FACILITY. (14) THT		5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. 15 Jun 79 — 31 Oct 80 (9)
		6. PERFORMING ORG. REPORT NUMBER: 80-53
7. AUTHOR(s): (10) Dr. Clinton P. Mah	(15)	8. CONTRACT OR GRANT NUMBER(s): F30602-79-C-0174
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Pattern Analysis and Recognition Corporation, 228 Liberty Plaza, Rome NY 13440	(16)	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F (17) 16 45941631
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDT), Griffiss AFB NY 13441	(11)	12. REPORT DATE: November 1980
		13. NUMBER OF PAGES: 70
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 70		15. SECURITY CLASS. (of this report): UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Zbigniew L. Pankowicz (IRDT)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Question Answering Methodology
Computational Linguistics
Relational Data Modeling
Multiple-Database Access
Natural Language Query Processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The report documents cumulative results of a 15-month R&D effort consisting in development of an adaptive query facility (AQF) for interactive exploitation of differently formatted target databases. AQF is a software package intended for experimentation and testing in operational environment. It provides a flexible and transparent on-line user access to target databases of arbitrary structure, and offers a variety of services including natural language query processing, relational modeling

(Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Item 20 (Cont'd)

cont. → of target data structures, correlation of data from various databases, and generation of informative report displays for users lacking experience with computers. AQF operates through an intermediate relational data model, thus insuring independence of the organization of target databases and their respective database management systems. Software package consists of a natural language query processor; a general target database access module; an automatic report generator; grammar and dictionary entry software; database mapping table software; diagnostic and validation tools, and a basic grammar for English query language. The package is written in FORTRAN for portability and runnable on small scale processors like DEC PDP 11/70 and PDP 11/45 under RSX-11M. AQF is also implementable on a micro-processor. Some commercially available CRT terminals can accommodate a 16-bit DEC LSI-11/03 micro-processor, up to 128K bytes of static MOS memory, serial interface, and a dual floppy-disk drive at a total price under $12,000. RSX-11M can run on such a micro-processor system, thus insuring immediate implementability of AQF in the present FORTRAN version on the system.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

DTIC
SELECTE
MAR 3 1981
D

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

EVALUATION

The objective of this effort consists in development of an advanced query facility (AQF) for experimentation and testing, in order to assess the utility of natural language access to on-line target databases in the operational environment.

Although AQF is conceptually similar to a few other query systems with natural language front-end, it differs from them in the following important respects: high degree of portability (AQF software is written in FORTRAN); operational economy (implementability on processors as small as DEC PDP-11/45); target database independence (AQF makes no aprioristic assumption about database structure), and multiple-database access capability that requires only a minimal additional programming. Furthermore, according to the Report, operation of AQF requires no special software other than a FORTRAN compiler at the level of DEC FORTRAN-IV-PLUS and a linker with overlaying for operation on processors with small address spaces.

It is also worth noting that AQF can be implemented at a low cost on a desk-top microcomputer, or on any off-the-shelf CRT terminal that can accommodate requisite microprocessing hardware components, such as those described in Section 7.2, USER DEVELOPMENT, pp 7-3 - 7-4.

According to the Report, "the main advantage of AQF is its applicability to existing data" and "the capability of generating information displays correlating data from different data bases." (cf. 7.1.2 Multiple Data Bases, p. 7-2)

Zbigniew L. Pankowicz

ZBIGNIEW L. PANKOWICZ
Project Engineer

1. INTRODUCTION

The Adaptive Query Facility (AQF) is a portable collection of data base access software developed by PAR Corporation under contract F30602-79-C-0174 for the Rome Air Development Center. The facility was designed primarily to address the problem of flexible, but transparent, user access to one or more existing online formatted data bases. AQF operates through an intermediate relational data model for data base independence and supports natural language queries. It is implemented in FORTRAN and can run on medium-scale processors like the DEC PDP-11/45 and PDP-11/70.

This report describes the various work carried out under the AQF contract. It also describes what AQF consists of, how the components of AQF operate and interact, and where AQF might be applied most effectively. Details of algorithms and their implementations are not described; these can be found in the AQT project final and in the AQF program documentation. A separate report, "The AQF User's Manual (System Manager's Implementation Guide)" describes in detail how to configure an AQF system for a specific target data base. In Section 4 of this report, there is a comparison of AQF with other data base access approaches.

1.1 AQF DESCRIPTION

AQF comprises a set of FORTRAN subroutines that may be incorporated into an online information system to provide an interactive query capability or that may be built into a stand-alone user interface to a target data base. In its full configuration, AQF is organized into five separate passes, patterned after the structure of a compiler for a high-level programming language.

The overall design of AQF aims at providing a simple, but flexible query capability oriented toward users with little computer training. There is no intent to demonstrate any sort of machine intelligence in AQF; but the ability of the different passes of AQF to support each other allows AQF to provide

capabilities usually associated with more complex natural language systems. The handling of linguistic reference in AQF is an example of this (see Section 2.2).

1.1.1 Design

Each of the five passes in full AQF performs a distinct function:

Pass 0 is a table-driven query language processor that rewrites a user query into an intermediate query referring to an internal logical model for a target data base. User queries may be in natural language, depending on the grammar driving the query processor.

Pass 1 is a linguistic reference resolution module for intermediate queries relative to an internal logical data model. This handles references that cannot be handled conveniently in the grammar for Pass 0. It distinguishes between anaphoric reference and co-reference.

Pass 2 converts a resolved intermediate query into a target data base access sequence. This process is table-driven to allow AQF to be independent of the target data bases. Some optimization is carried out.

Pass 3 interprets target data base access sequences to search for and retrieve data. It creates lists of retrieved items that may be referenced in following queries (co-reference).

Pass 4 formats lists of retrieved items for display. Information not actually specified in a data access sequence may be included when needed for effective interpretation of displays. This is table-driven.

Pass 5 actually puts up an AQF output display, with an optional sorting of output lines by fields.

The five passes are modular. Pass 3 is in a sense the heart of AQF. Depending on the circumstances, the other passes can be replaced or dropped

entirely.

In addition to the five passes, AQF includes various support programs to create and to maintain the various tables driving the passes. All AQF tables are in binary form, but they are generated from ASCII text files. The programs for doing this range in complexity from a large grammar entry program to programs that convert ASCII straight into binary. All support programs are written in FORTRAN, like the rest of AQF.

1.1.2 Services

In its full configuration, AQF offers a variety of services to help a user in the analysis of stored data. To begin with, AQF logical data models lets the user see the structure of the target data base as a more simple hierarchy of relations with conventions to make names correspond to natural language usage. AQF itself can automatically translate references to an AQF logical model into references to target data bases. Furthermore, the AQF query processor combined with an English grammar allows even logical data models to become transparent by automatically translating input queries into intermediate queries. This also eleminates the need to learn any formal query language syntax.

The AQF search and retrieval algorithm together with the AQF report generator allows for correlation of data from different data bases, even though they might have disparate structures. This provides a capability similar to the JOIN operator of relational data base systems, but completely transparent to user. AQF in effect can automatically construct data base relations in response to an information request.

AQF can be applied to any target data base with only a little additional programming. Such programming is unavoidable because AQF cannot anticipate all possible data types and all possible data access methods, but this is isolated to a small number of submodules called by the various passes of AQF.

Use of AQF requires no special software other than a FORTRAN compiler at the level of DEC FORTRAN-IV-PLUS and a linker with overlaying when running on processors with small address spaces.

1.1.3 Examples

The following sample of AQF usage is drawn from a stand alone demonstration system having a Soviet fighter data base implemented with FORTRAN files. This illustrates what AQF can do from the user's standpoint. What AQF will actually do in an operational application will depend on the query language vocabulary defined and the degree that the report generation options are employed.

```
$ RUN QRYD

* A  Q  F   DEMONSTRATION  SYSTEM *
*       VERSION   08-06-00        *


QRY>TELL ME ABOUT AIRCRAFT.


AIRCRAFT(?)
(,)


@1 AIRCRAFT
THIS IS A SOVIET FIGHTER AIRCRAFT DATA BASE, DERIVED FROM
UNCLASSIFIED PUBLICATIONS.  IT DESCRIBES CREWS, FUSELAGES,
WINGS, ENGINES, ARMAMENTS, AND PERFORMANCE.


QRY>LIST FIGHTERS ORDERED BY NATO NAME.


AIRCRAFT(?)[ROLE:=(AW)F(GA)(S)(/B)(/T)(/A)]
, IDENTIFICATION(?)[/NATO/ NAME:=*'S]
(,)



ROLE     /NATO/ NAME
-------------------------------------
F/B      FARMER       MIG-19
F/B      FARMER-C     MIG-19SF
F/B      FARMER-D     MIG-19PF
F/B      FARMER-D     MIG-19PM
FGA/T    FISHBED      MIG-21
FGA/T    FISHBED-C    MIG-21F
FGA/T    FISHBED-D    MIG-21PF
FGA/T    FISHBED-F    MIG-21PFM
FGA/T    FISHBED-J    MIG-21PFMA
FGA/T    FISHBED-J    MIG-21MF
FGA/T    FISHBED-K    MIG-21MF
FGA/T    FISHBED-L    MIG-21MF
AWF      FISHPOT      SU-9
FS/A     FLOGGER      MIG-23
FS/A     FLOGGER-B    MIG-23B
FS/A     FLOGGER-C    MIG-23U
F/B      FRESCO       MIG-17
```

```
QRY>WHAT ARE THE DIMENSIONS OF THE FARMER?

AIRCRAFT IDENTIFICATION(1!)[NATO NAME!=FARMER]
, DIMENSION(?)
(,)



                                          -FT----------FT----------FT
MIG-19       FARMER       LN=  48.88   HT=  13.19   WS=  29.54

QRY>HOW MUCH DOES IT WEIGH?

.
. AIRCRAFT WEIGHT(?)[..!=*]
(,)


                              WEIGHT
                              ------LB
MIG-19       FARMER            19180

QRY>PRINT THE ARMAMENT OF THE FARMER.

AIRCRAFT IDENTIFICATION(1!)[NATO NAME!=FARMER]
, ARMAMENT(?)
(,)


-------------------------------------------------------------------
MIG-19       FARMER       *CFGN*     1 AA MISSILE  AKALI
                                     3    CANNON
                          *CFGN*     4 AS ROCKET
                                     2    BOMB
                                     3    CANNON
                          *CFGN*    -1 AS ROCKET
                                     2    BOMB
                                     3    CANNON
```

```
QRY>HOW FAST CAN IT FLY?

.
. AIRCRAFT PERFORMANCE
, PERFORMANCE(?)[SPEED:=*]
(,)
```

		SPEED	
		MACH	FT
MIG-19	FARMER	0.50	80000
		0.60	70000
		0.75	60000
		0.97	50000
		1.24	40000
		1.32	30000
		1.09	20000
		0.84	10000
		0.66	0

```
QRY>THE FRESCO?

. AIRCRAFT IDENTIFICATION(?)(1!)[NATO NAME:=FRESCO]
(,)
```

		SPEED	
		MACH	FT
MIG-17	FRESCO	0.39	40000
		0.48	30000
		0.62	20000
		0.80	10000
		0.92	0

```
QRY>GIVE WEIGHTS OF FIGHTERS WITH COMBAT RADIUS OVER 500.

AIRCRAFT[ROLE:=(AW)F(GA)(S)(/B)(/T)(/A)]
, PERFORMANCE[/COMBAT/ RADIUS:>500]
, WEIGHT(?)[,,:=*]
(,)
```

ROLE			/COMBAT/ RADIUS
			N MI
FS/A	MIG-23U	FLOGGER-C	600
FS/A	MIG-23B	FLOGGER-B	600
FS/A	MIG-23	FLOGGER	600
FGA/T	MIG-21PFMA	FISHBED-J	1183
FGA/T	MIG-21PFM	FISHBED-F	1183
FGA/T	MIG-21PF	FISHBED-D	1183
FGA/T	MIG-21MF	FISHBED-L	1183
FGA/T	MIG-21MF	FISHBED-K	1183
FGA/T	MIG-21MF	FISHBED-J	1183
FGA/T	MIG-21F	FISHBED-C	1183
FGA/T	MIG-21	FISHBED	1183

ROLE			WEIGHT
			LB
FS/A	MIG-23U	FLOGGER-C	28000
FS/A	MIG-23B	FLOGGER-B	28000
FS/A	MIG-23	FLOGGER	28000
FGA/T	MIG-21PFMA	FISHBED-J	27750
FGA/T	MIG-21PFM	FISHBED-F	27750
FGA/T	MIG-21PF	FISHBED-D	27750
FGA/T	MIG-21MF	FISHBED-L	27750
FGA/T	MIG-21MF	FISHBED-K	27750
FGA/T	MIG-21MF	FISHBED-J	27750
FGA/T	MIG-21F	FISHBED-C	27750
FGA/T	MIG-21	FISHBED	27750

```
QRY>^Z

BYE...
```

1.2 BACKGROUND

Online data bases allow for fast responses to information requests and make the data handling capabilities of computers available to users. The actual usefulness of online data bases, however, will depend a great deal on how well a user and a system can communicate. In many situations, this may be difficult; for example, it might require the user to learn a private access language and to be able to designate items of data in a particular way. Typically this language will have a logical form, and the data will be organized according to global systems constraints -- making the data base opaque to a user who is neither a logician or a data base expert. A system also may have problems communicating information back to the user in a way that is readily interpreted.

An additional complication is the frequent existence of multiple data bases, which for historical reasons were developed separately with their own private access languages and data base organizations. This multiplies the difficulties for a user and makes it unlikely that the user will be able to make connections between data stored in different places. How to display such data in a way that makes sense to a user is also not obvious, especially if the connections are not a simple direct one-to-one relationship.

One important approach to improving communication between user and system is the relational data model [3]. This approach defines data base access in terms of an abstract model of stored data, where details not of interest to the user are masked. Because there is much less to talk about, a private access language can become simpler, and the designation of data items is much easier. The problem of communication, however, has not gone away. In the typical relational system, the nonexpert user still faces a private access language in logical form and finds data organized primary according global systems constraints. Furthermore, there is still the question of how to impose a relational model effectively on existing data bases with different kinds of structures.

1.2.1 History

The development of AQF began with a basic observation about the designation of data in REL [7] system, an online interactive data analysis system based on a binary relational model. In a proposed application of REL to a Soviet aircraft data base, the names of relations tended to be rather long and to contain many words in common; for example, "AIRCRAFT DESIGNATION", "AIRCRAFT TYPE", "(AIRCRAFT) FUSELAGE LENGTH". This suggested that the relation names might be factored out, yielding a hierarchical structure that was generalized into the notion of a hierarchy of n-ary relations, each designated by a single word (see Section 2.3).

The use of a relational hierarchy as a logical model made it fairly easy to designate data in a query language because the factoring out of names simplified the construction of a dictionary. This also made it easier to construct a query language based on English syntax, since the semantics of the query language would be limited to aspects of a relational hierarchy; that is, a query has to map into references to relations, fields, functions, or values, no matter how complicated it might be.

The major question of feasibility in this general scheme was whether there existed a reason for automatically mapping a relational hierarchy logical model into existing target data bases of arbitrary structure. This matter was addressed in the Advanced Query Techniques (AQT) effort [5], in which the basic algorithms now in Pass 1,2,3, and 4 of AQF were developed and demonstrated in a system running with a Soviet fighter data base. This showed that it was possible on a DEC PDP-11/70 to have a query facility in which the structure of target data bases (as well as logical data models) could be made almost transparent.

The Advanced Query Facility effort grew directly from AQT. Its goal was to develop a portable data base access facility out of the AQT demonstration system. This involved the extensions and improvements of passes 1 through 4,

the implementation of pass 0 in FORTRAN, and the addition of pass 5, a sort module. The overall structure of AQF was laid out to keep target data base dependence to a minimum. Everything now has been brought together into one software package available for distribution on a small (600 ft.) reel of magnetic tape. Two AQF demonstration systems have been implemented, one with the AQT Soviet fighter data base and the other with a U.S. airfield data implemented with DEC RMS-11 data base software.

1.3.3 Related Systems

AQF is similar in concept to a number of other systems currently under development: PLANES [8], LADDER [4], and others. These represent direct application of natural language processing techniques to data base access where the system developer has no control over the content of the data base. The content is determined by a user with a specific end in mind; and the system is required both to accommodate initially given data and to adapt to any new data as user requirements change.

This is different from laboratory natural language systems (e.g. SHRDLU [9]) where the planned capabilities of a system usually determine the choice of a data base. Such a system tends to be so highly tailored to its data base that it cannot be readily applied to a new one. It serves primarily as a vehicle for investigating and demonstrating new natural language techniques; it is not intended as a practical tool for actual users.

AQF and related systems have the opposite emphasis. These are supposed to help users to overcome real information problems and need not involve any new natural language techniques at all (although this almost always is still so). The difference in emphasis means that simplicity, speed, reliability, and understandability of a system are at least as important as its technical capabilities, if not more so.

AQF differs from other natural language systems for applications in three key ways:

- It is a self-contained package written in FORTRAN and developed to run on medium-scale minicomputers.
- It makes no assumption about the structure of a target data base or about the data base manage system; multiple data bases present no difficulty.
- It incorporates extensive report generation capabilities as well as query processing capabilities.

AQF in short is designed to be applicable in almost any situation where data base access might be a problem. It does not require any particular hardware or software to run.

1.2.3 Target Applications

Although AQF can run on large mainframe processors, it is most advantageous on mini-computers. It was originally designed to serve users such as intelligence analysts needing a dedicated processor to work with sensitive data. Typically this rules out large mainframes, and with smaller machines, the user has to work with data base systems having only primitive interactive query facilities, if any at all. Given in addition that users would tend to be inexperienced with computers, bringing a natural language query capability to smaller machines seems a good idea.

Natural language data base access is helpful when a user has the problem of correlating data from different places in order to perform a task. This kind of situation is difficult for a menu-driven or similar fixed-query system to handle because these imply fixed displays of information that are not always presenting exactly what the user wants. Accessing more than one data base at a time is a special case of this correlation problem.

The AQF software package aims at supporting users, expert or not, who want to manipulate online data in various ways to discover possible connections. Its natural query language front end, its general retrieval mechanism, and its report generation capabilities sent to provide flexible access when needed for analysis of data from different sources. It also eliminates the need for any special user interface on data base systems.

2. CONCEPTS

The key concepts underlying AQF have already been introduced informally in Section 1. This section will define these more rigorously in order to provide a theoretical basis for subsequent discussion. Discussion will be in the following order: (1) natural language, (2) intermediate queries, (3) relational hierarchies, (4) table-driven data base mapping, (5) access paths, and (6) semantic dependence of data.

2.1 NATURAL LANGUAGE

No one yet knows how to program a computer to understand language in all the ways that person can: reading a magazine, carrying on a conversation, listening to a radio commercial, and so forth. This does not mean, however, that natural language is impractical for computers. In any practical computer application, there is never any question of completely emulating human language behavior; rather the need is that of identifying the kinds of transactions between person and machine that have to take place and setting up conventions to make this process as painless as possible.

Natural language provides a particularly good source of potential conventions for transactions because it represents a highly evolved tool for communication and is something that people can use skillfully without conscious effort. To build a natural language computer interface, the strategy is straightforward: first, make no query language conventions that conflict with natural language; and second, incorporate at least counterparts to the basic referential conventions of natural language, including the contextual replacement of long linguistic expressions by short ones. The first half of the strategy avoids making a user unlearn things; the second introduces at least the most developmentally primitive part of natural language (cf. baby talk) into a query language.

For a data access domain of discourse, not much more than this is actually required for natural language. In the case of AQF, the semantics of a query language is established by a relational hierarchy logical model; anything meaningful must refer to some aspect of that model: relations, fields, values, or functions. Syntax is relatively unimportant here, for no matter how complex all meaningful syntactic relationships expressed in any query have to be interpretable ultimately in terms of the hierarchical structure of the logical model.

This enormous simplification of the natural language problem of course entails a certain cost; "how" and "why" queries cannot be handled conveniently, for example. Nevertheless, the query language that can be defined is adequate enough for general data access, and there are certainly implementational advantages to simplicity. A query language with relational hierarchy semantics is in any event more natural than one based on predicate calculus disguised to look like English by writing quantifiers and connectives out as words.

2.2 RELATIONAL HIERARCHIES

A relational hierarchy can be thought of as a special kind of relational data structure. Formally, it is simply a collection of the usual sorts of n-ary set-theoretic relations defined over various classes of data objects with a partial ordering imposed on the relations. The partial ordering, is defined linguistically as follows:

1. each relation has a name consisting of a single common noun; e.g. AIRCRAFT.

2. if a field is defined for a relation, then standard linguistic usage permits the relation name to precede the name of field as a modifer; e.g. AIRCRAFT NATO NAME for the field NATO NAME.

3. any completely ordered sequence of relation names is linguistically acceptable; e.g. AIRCRAFT, AIRCRAFT ENGINE IDENTIFICATION, ENGINE IDENTIFICATION.

4. each field is as high up in the relational hierarchy as possible, consistent with the sense of the field name and linguistic usage implied by the preceding requirements.

Here is a simple example of a relational hierarchy.

```
AIRCRAFT
    .
    . . . IDENTIFICATION [NATO NAME, SERVICE NAME]
    .
    . . . WING
    .       .
    .       . . . DIMENSION [SPAN]
    .
    . . . FUSELAGE
    .       .
    .       . . . DIMENSION [LENGTH]
    .
    . . . ENGINE [TYPE]
    .
    . . . IDENTIFICATION [MANUFACTURER, DESIGNATION]
```

The point of a relational hierarchy is to take advantage of the many degrees of freedom possible in the definition of a relational data model so as to define a model conducive to natural language. The form of relational hierarchy represents in fact an attempt to make the designation of data elements in a model correspond closely to linguistic usage regarding words modifying other words. Data dependence in the model in some sense is made to parallel linguistic dependence in a natural query language.

Although this approach may seem simplistic as far as natural language systems usually go, it actually works out fairly well in the specialized data base access application that AQF is intended for. AQF does not really need to have anything more elaborate, and from the standpoint of designing for maximum

portability, it should not. Relational hierarchies allow AQF to get by with relatively little front end query processing.

2.3 INTERMEDIATE QUERIES

The first part of AQF query processing is to translate an incoming natural language query string into a formal intermediate query string referring to a relational hierarchy logical model of a target data base. This approach has three important advantages: it improves the modularity of the query processor, the intermediate forms make it easier to deal with contextual relationships between queries, and the intermediate query string can be displayed to show a user whether an input query was processed correctly.

An intermediate query formally consists of a series of clauses having the following form

```
xxx yyy zzz [AAA := TT, BBB :> UU]
```

where xxx yyy zzz is a sequence of consecutive relation names and the brackets enclose an optional list of conditions on values as associated with fields of the rightmost relation. A given clause may be dependent or independent according to its contextual aspects.

A dependent clause is one that can be interpreted only in the context of the preceding clause; in the intermediate query syntax, a dependent clause is marked by a "." character at the start of the clause. The sequence of relation names in dependent clauses need not start from the top level of a relational hierarchy. An independent clause in contrast can be interpreted absolutely without regard to context, and its sequence of relation names must start from the top level of a relational hierarchy.

An intermediate query consists of a sequence of clauses, with the first clause either independent or dependent and following clauses all dependent. The relation name sequence must be extendable to the top of a hierarchy in a way consistent with preceding clauses up to the first independent clause; the sequence for a dependent clause must start with a relation name that is either the same as one occurring in the sequence for a preceding clause or immediately below one with the respect to the relation hierarchy.

Individual relation names in a sequence for a query clause may be marked according to information to be returned if the conditions expressed in a query can be satisfied in a target data base. Three markings are defined for AQF: "(?)" to request retrieval of field values, (Y/N?) for a simple yes or no response, and "(#?)" for a count of matching instances. In addition, a relation name can be marked with a count specifier of the form (n!), where n is number when a specific count of matching instances is required or is omitted entirely to indicate specificity of reference without a definite count.

An intermediate query is terminated by one of three different possible markers: "(.)" denoting simple termination, "(&)" denoting termination with expectation of a following intermediate query related by a logical AND, and "(¦)" denoting termination with a following query related by logical OR following. These pertain to the possible combination of results meeting different query conditions to make a single information display.

Here are some examples of intermediate queries based on the relational hierarchy example in Section 2.2.

```
AIRCRAFT (?) IDENTIFICATION [NATO NAME := *].
(.)
"GIVE NATO NAMES OF AIRCRAFT."

AIRCRAFT (#?)

. WING DIMENSION [SPAN :> 10 M]
(.)
"HOW MANY AIRCRAFT WITH WING SPAN OVER 10 METERS?"

. DIMENSION [SPAN :> 12 M]
(.)
"HOW MANY OVER 12 METERS?"

AIRCRAFT (Y/N?)

. WING DIMENSION [SPAN :> 10 M]
(|)
. FUSELAGE DIMENSION [LENGTH :> 15 M]
(.)
"ARE ANY AIRCRAFT WITH WING SPAN OVER 10 METERS OR
LENGTH OVER 15 METERS?"
```

The "*" value is a special "wildcard" that matches any defined value for a field. By convention, values for all fields marked by "*" will be returned when a query clause is marked with a "(?)".

Intermediate queries do not actually constitute a semantic formalism for AQF in the sense of unambiguously specifying meaning. They are not yet completely resolved contextually; this job is left for later stages of AQF so as to lighten the task of the AQF parser, which must translate natural language into intermediate query forms. The natural language portion of AQF could also be dropped entirely, witn the user either entering intermediate queries directly or possibly employing some kind of interface software to generate them indirectly.

2.4 TARGET DATA BASE MAPPING

This is the heart of AQF. The capability of mapping a relational hierarchy onto a target data base of arbitrary structure makes it possible for query language processing to be developed without regard to any data base and in the end provides AQF with its adaptability. The AQF mapping is a table-driven procedure based on establishing connections between fields in a relational hierarchy and data items in a target data base. This provides a way of translating semantic dependence between designated fields of a query into actual data access linkages in the target data base.

In a sense, AQF is an expert on data base structure. Its mapping tables and related access procedures constitute an overall description of a target data base as needed for general interactive user access. Altogether, there are three main tables, two main linkage procedures, and a special table and procedure for indexed access.

- o A field correspondence table associates a named field of a relation with a data item of a particular record type within a target data base. This includes information on data type and units of measurement.

- o An intra-relational link table describes the various target data access linkages that tie together the fields associated with a given relation. This handles the case of a relation encompassing data from several target data base record types connected by 1-to-1 linkages.

- o An inter-relational link table describes the various target data access linkages corresponding to the hierarchical connections between the relations of a logical data model.

- o The "first record" and "multiple" records procedures in AQF define operationally the types of target data access linkages referenced in AQF mapping tables. The first record procedure follows a link to get

the first record instance pointed to; the multiple records procedure gets succeeding records after the first in case of a 1-to-many link. Linkage procedures have to be tailored to a given target data base because it is impossible to anticipate all possible linkages in AQF.

- For indexed access, an indexed fields table specifies the fields that are listed in an index, and an associated procedure contains code for using the various available indexes. Indexed access is treated separately because it only applies only to the initial part of access to a target data base.

The AQF mapping tables and procedures in general will not include all possible linkages in a target data base. This is in part because working from a relational hierarchy makes only 1-to-1 and 1-to-many linkages of interest in the target data base. It is also usually desirable for security and other reasons to limit the access of any given interactive data base user with logical data models map onto the portion of a data base needed by the user and no more.

2.5 ACCESS PATHS

An AQF access path is a branching sequence of target data references, each except the first being related to the preceding by a single target data base access linkage. Access paths are represented internally by AQF as tree structures. They are used by the AQF search and retrieval module in traversing a target data base to find those data fields for which search conditions must be checked or for which values are requested in a query.

AQF sees a target data base as a discrete two-coordinate "access space", with the first coordinate being a target data base record type and the second being a logical mode relation through which data was referred to originally (the retention of the identity of relations is needed for report generation later.) The idea is to have each segment of an AQF access path show how to get

from one point of the access space to another.

An access path is constructed from an intermediate query by the following procedure applied to each field specified in the query:

- o Look up the field in the field correspondence table to get a target record type for it. This locates the field specification within access space.

- o Use the inter-relational link table to move through access space in a direction that corresponds to moving up in the relational hierarchy.

- o If the preceding fails, use the intra-relational table to move to another point in access space and try again.

- o Stop upon reaching a point in access space which can be reached by a top-level access method, usually sequential or indexed. If this is impossible , then the mapping tables are in error.

The access sequences for all fields in an intermediate query are finally merged to get a single access path.

2.6 SEMANTIC DEPENDENCE

In order to generate a user interpretable display of retrieved data, AQF has to keep track of the semantic dependence of data items, which is usually not apparent from looking at a target data base alone. Such dependence is defined by a data access path, in which is implicit the relational hierarchy serving as the logical model for user access. The AQF usage of access paths makes fields occurring along a path depend on all fields preceding it on the path.

This dependence is the basis for AQF report generation. The idea is to think of the fields specified along a branch of an access path as a kind of composed data relation, not unlike a new relation formed from other relations through set-theoretic operations like JOIN, INTERSECTION, and RESTRICTION. The AQF report generator simply produces a display of each of the composed relations defined by an access path, taking the designated field values from retrieved record instances for the path.

The basic AQF display procedure is augmented by what is here termed as "mandatory key fields." These are fields that are important to show when displaying a composed relation, but that are not always explicitly requested in a query. AQF is set up to include these key fields automatically, working from information in an AQF mandatory fields table. This table associates implicitly requested fields with particular points in an access space so that these fields can be inserted automatically into an access path where it crosses those points.

Typically the mandatory fields table includes the primary keys for a target data base since these by definition serve to identify items of stored target data. In a relational hierarchy model, those key fields will tend to be in the upper part of a hierarchy because most other data fields will be semantically dependent on them and thus be below them. Special non-data fields required only for marking off displays can also go into the mandatory fields table; these would contain no information, serving only to highlight certain groupings of retrieved data.

3. ORGANIZATION

This section will look at AQF from an implementational standpoint. The overall structure of AQF is like that of a multi-pass compiler for a programming language. The multi-pass approach works out well when integrating many diverse algorithms and it is a necessity where limited address space prevents having everything in main memory. The discussion of AQF here will be split up along these lines as well.

3.1 QUERY LANGUAGE PROCESSOR

Because the AQF Query Language processor is table-driven, it theoretically can be set up to handle any kind of query language; but for the most part, it is tuned for subsets of English. The query language processor in Pass 0 has three principal components: a word stemmer, a sentence parser, and a rewriting module.

3.1.1 Lexical Analyzer

To avoid a query language dictionary having to list all simple inflectional variants of every word, AQF incorporates a -s, -ed, and -ing suffix remover to get the root forms of words. This currently recognizes over 400 patterns of word endings and is able to restore final "e" on words where it has been dropped and to undouble final consonants where necessary.

In addition to the stemmer, AQF lexical analysis also includes special procedures for handling numbers, unknown words, and multi-word lexical items. The treatment of unknown words in query language processing is particularly important because it is often inconvenient to include all the possible values for a data field in a dictionary for the language; the basic AQF query language grammar provides several mechanisms for interpreting unknown words as literal values when they are associated with field names in a query.

3.1.2 Parsing

The AQF parser is essentially the same as that described in the Advanced Query Techniques final report [5]. It is a table-driven bottom-up parser built up on the framework of Vaughan Pratt's implementation of the Cocke-Kasami-Younger algorithm for parsing context-free languages [6]. The parser has been enhanced for natural language application with the inclusion of syntactic and semantic features, which are extensions in the direction of Van Wijngaarten grammars; these allow the syntax of a query to be expressed with a much smaller number of rules than in the case of a straight context-free grammar.

The parser also makes special provision for right and left recursion in parsing. Phrases that are absorbed into a larger phrase of the exact same type by application of a right-recursive rule are eliminated from further consideration, saving the effort of following any more syntactic consequences for them. Left-recursion is recognized in a similar fashion, but only for the case of rules relating the root form of a word and its inflectional endings.

The AQF parser is written entirely in FORTRAN, a reimplementation of the assembly language parser used in the AQT demonstration system. It is actually smaller in its overall space requirement than the AQT version because the use of external dictionary files in AQF allows for smaller internal tables. The parser is organized to be able to run easily within the address space of processors like the DEC PDP-11/45.

3.1.3 Text Editor Semantics

The AQF parser produces a syntactic analysis of a query, describing the rules of grammar applying to the query and giving the definitions of the words in the query. This information is then used to rewrite the input query into an intermediate query form. The procedures for doing this are incorporated in the rules of grammar used for analysis and the definitions for words.

In AQF, the semantics of any query constituent is defined as a procedure expressed in a special language for string manipulation. This language is best described as a block-structured text-editor language, consisting of the basic operations associated with an interactive text editor, like INSERT, FIND, or DELETE, combined with structured programming control structures, like IF-THEN-ELSE or DO-WHILE. The language implements recursion with both global and local variables and provides for dynamic allocation of storage.

The AQF semantic language for query processing also provides for shared access to local variables defined in procedures. These allow semantic procedures to have some control over the execution of other semantic procedures, letting the rewriting process for intermediate queries be context-sensitive even though parsing remains on a context-free basis.

3.2 REFERENCE RESOLUTION

For some natural language systems, reference tends to be an extremely difficult problem. In AQF, the situation is simpler because the use of relational hierarchies to define the meaning of queries severely limits the possibilities for reference. The resolution of query references in AQF is done in two stages: aspects involving only contextual substitution of words are handled almost entirely in the rewriting component of the query processor; more complex aspects are handled on Pass 1, which takes immediate queries as input.

3.2.1 Anaphoric Reference and Coreference

Reference resolution in AQF consists of associating all references in an intermediate query with specific elements of a relational hierarchy. This involves two tasks in Pass 1: first, the intermediate query string from Pass 0 must be converted into a tree structure where any semantic dependence implicit in the ordering of clauses is made explicit; second, any reference to preceding query must be clarified as either co-reference or anaphoric

reference.

For AQF, co-reference means that a query is referring to the same things that a previous query referred to. This is handled by restricting a search to what was retrieved before. Anaphoric reference, on the other hand, makes no such restriction; it requires rather that parts of a preceding query be copied over to the current query in order to fill it out. For example, co-reference with two intermediate queries

```
AIRCRAFT IDENTIFICATION [NATO NAME := FOXBAT]
.WING DIMENSION (?) [SPAN := *]
(.)
. AIRCRAFT FUSELAGE DIMENSION (?) [LENGTH := *]
(.)
```

as opposed to anaphoric reference

```
AIRCRAFT IDENTIFICATION [NATO NAME: = FOXBAT]
. WING DIMENSION (?) [SPAN: = *]
(.)
. AIRCRAFT IDENTIFICATION [NATO NAME: = FLOGGER]
(.)
```

In the anaphoric case, AQF must retrieve a completely different set of target data base records in order to respond to the query.

Pass 1 of AQF determines the type of reference intended in a query by comparing its field references with those of the resolved tree form of the

preceding query. This is to see whether any value associated with a field has changed and whether any new fields or relations have been introduced. Co-reference is assumed unless the preceding query is superseded in some way. This scheme is extremely simple compared to reference in most natural language systems, but seems to work well enough for intended AQF applications.

3.2.3 Ellipsis

Most natural language systems devote much effort to handling query sequences of the following sort:

> What is the length of the Foxbat?
> Of the Foxbat?

The second query is a problem because it is a sentence fragment where a full sentence is wanted. A special mechanism is needed to expand the second query into full form for processing, with complications for a parsing scheme.

AQF is unusual in that it has no problem with elliptical queries at all. Because AQF query processing primarily is looking for relations, fields, and values, sentence fragments present no more difficulty than full sentences. Furthermore, the break down of intermediate queries into clauses makes ellipsis transparent at that stage of processing. No new algorithms are needed.

3.3 ACCESS PATH GENERATION

Pass 2 of AQF generates a target data access path from a resolved intermediate query. This is done by mapping each field reference in the intermediate query into an item in the target data base, deriving an access sequence for each item, and then merging these to get the semantic dependence between referenced data items. The process amounts to imposing the semantic

relationships defined in a logical data model onto target data.

3.3.1 Target Data Linkage

AQF assumes that target data is organized into various types of records. These may be data aggregations of arbitrary type, but usually will be contiguous allocation of data storage with data items of different types defined to begin at various offsets within a record. Extraordinary record types, such as ones involving non-contiguous allocations or overlapping allocations, must be handled by inclusion of special procedures in the AQF access subroutines, which define access linkages specific to a target data base.

The simplest target data linkage is the trivial one when two dependent data items occur in the same record type. In general, however, data will be in different record types not even directly linked, so that to get from one item to another may be fairly complex; this may involve scanning of records sequentially, following various kinds of pointers and list links, hashing secondary keys, or perhaps tracing some highly exotic linkage unique to given target data bases. For example, the linkage between data in two different data bases may consist of extracting a key value from one data base and transforming it to an analogous key value in the other.

There is considerable flexibility in defining data linkages for a particular AQF application. Not all target data base links of course have to be made known to AQF. In addition, data base links do not even have to conform strictly to a relational hierarchy. For example, instead of following the shortest path between two points in access space, one can insert a detour, which may involve data not encompassed by a logical model or, for that matter, virtual data not even in a target data base. This is useful for report generation, described below in section 3.5.

4.4.2 Field Name Look-up

Pass 2 expands field name references in a resolved intermediate query before looking them up. This serves mainly to simplify a problem in parsing field references where certain words have possibly ambiguous usage; for example, the word "MAXIMUM" in "MAXIMUM LENGTH" could either be actually part of a two-word field name or be the specification of a function to be applied to a field called "LENGTH." This matter could have been addressed earlier in Pass 1 as a disambiguation problem, but it is easier to deal with in Pass 2, where information on target data base mapping is available.

Beside recognizing function specifications, AQF uses a special sub-key table in the expansion of field names in order to identify contexts when certain words may designate values to select one of several similarly named fields to process. This is a more complicated situation than the case of functions, because it generally involves an additional target data item and producing the appropriate links to it. Sub-keys are helpful to define, however, because they simplify a query language grammar.

4.4.3 Indexed Access

Indexing applies only to the start of a data access sequence when a specific value has been given for an indexed field, identifiable through an AQF table for such fields. When indexed access is possible on a field, then that is incorporated by Pass 2 to start an access path; otherwise, AQF generates an access sequence for the field in the normal way. Only one use of indexed access is allowed per access path.

Actual indexing methods have to be supplied to AQF as part of an indexing subroutine "INDEX" specific to a target data base. In cases where an index is always on a data item in a preceding record type, and thus not at the start of an access sequence, the indexed access is treated as merely another kind of linkage in the AQF target data linkage subroutines.

3.3.4 Special Access Methods

AQF refers to target data in terms of a given record number of a given record type plus a variable offset displacement for dealing with arrays of values. All special access methods are encoded in the AQF linkage subroutines (FIRSTR, MLTPSR), which take a current instance of a target data base record and a linkage type as arguments and returns a record number and array displacement value for the record type linked to. An AQF record access subroutine tailored for a target data (ACCESS) base serves to read in a particular record instance given its number and type.

All array data must have array limits and array element sizes defined in linkage procedures. Text data, such as a comment field in a target data record, is best treated as a special case of array data with array element size equal to some fixed output line size. This, however, permits no output buffering and no formatting to avoid breaking up of words across output lines.

3.4 DATA BASE SEARCHING

The AQF target data base search procedure in pass 3 is comparable in complexity to the AQF parsing algorithm. It is probably the most important component in AQF because it had to be developed before anything else could work. The search procedure is responsible for the bookkeeping that underlies the correlation of data from different places (possibly different data bases) and the maintenance of a context for interpreting co-reference.

The pass 3 search algorithm keeps track of matching record instances for a data access sequence by saving them in lists according to data record type and relation referenced through. A record instance is added to a list when it matches a search condition on an access path, and it is deleted if it fails to match another search condition. Deletion of a record instance from a list may require purging of all semantic dependent record instances from other lists; for this reason, record instances are also linked across lists to retain

information about their access sequence relationships.

3.4.1 Pattern Matching

AQF has a special pattern matching procedure for string data. It implements matching of optional substrings, of initial substrings only, and of alternative patterns against data items in a target data base. This serves primarily to make retrieval on classes of strings convenient, but can also be applied to search for occurrences of words in text fields of target data records. The latter capability would be useful for target data bases where "comment" fields of free format text tend to contain more information than the highly formatted portions of data records. This seems to be a fruitful area for further development.

3.4.2 Purging Precedure

The AQF purging procedure in pass 3 makes it possible for data base searches to be directed at results of preceding searches. The procedure was originally implemented to let search conditions on one branch of a merged access path further restrict the subsets of record instances matched for an access sequence previously followed. This was needed to handle what is essentially the co-reference problem within a single query; it also proved applicable to the problem of co-reference across queries.

In a successful AQF search, a matching record instance must be found for each point along a data access sequence. Different parts of a query, however, may refer to the same record instance, in effect applying multiple search conditions on it. If a second search condition eliminates a record instance from consideration, then it also eliminates all other record instances related to it along the data access sequence originally matching and bringing them together. This is because record instances are not retrieved individually but always as part of a semantically dependent set of record instances. In effect, each data access sequence defines a relational data n-tuple that has

to be treated as a unit.

Pass 3 of AQF maintains the internal links necessary to find all those semantically dependent record instances collected along a data access sequence. This is updated with each new record instance matched; upon the deletion of a record instance, all record instances related to it will be purged, and the internal links revised accordingly. The entire operation is completely invisible outside of pass 3.

3.4.3 Multiples of a Record Type

The occurrence of one-to-many data access links in a merged access sequence slightly complicates pass 3. Because AQF searches are depth-first along an access sequence, it is necessary to back up along a sequence to look for possible multiples of a record type after a search fails or successfully reaches the end of the access sequence. Furthermore, multiple record instances that meet search conditions must be saved along with internal links for dealing with co-reference.

AQF treats multiple record instances as each corresponding to separate relational data n-tuples for an access sequence. As long as any one of the multiples meets search conditions, then the sequence can be satisfied. The pass 3 purge procedure keeps track of multiples collected at each point of an access sequence to determine when all have failed to match and to initiate a purge of semantically dependent record instances then.

Searches involving values stored in arrays are handled in AQF as a special kind of multiple. This involves listing the same record instance more than once for at a point in a data access sequence, but with different array displacements to identify individual array elements. Maintenance of record instances and internal links would be as in the case of normal multiples.

3.5 REPORT GENERATION

The internal links established by Pass 3 for retrieved record instances serve as the basis for report generation in Pass 4. The idea is to display a table of selected data items from record instances matched along a data access sequence, looking at it as defining a kind of relational data n-tuple. Items are selected for display either by being explicitly asked for in a query or implicitly from mandatory fields that have to appear in a response to make it interpretable. Displays will be generated for each data access sequence retrieving record instances with an explicitly requested field, until all such fields have been taken care of.

Report generation comprises the largest amount of code in any of the passes of AQF. It provides a flexible way of bringing data from diverse sources together in the manner of data manipulation with a relational algebraic language. Queries of a yes/no or how-many type give the AQF user the additional option of ascertaining the existence or extent of retrieved data before producing a display. Queries specifying no fields at all have special significance; they are by convention interpreted by AQF as meta-queries requesting online documentation.

3.5.1 Mandatory Fields

A straight dump of data requested in a query often fails to provide enough information for proper interpretation. For example, asking for the "WEIGHT OF SOVIET INTERCEPTORS" should strictly yield a list of numbers, a situation like having the scores for baseball games without knowing the teams playing. To make output more meaningful, AQF puts three kinds of mandatory fields into displays besides explicitly requested data:

1. primary keys for target data records, since these can uniquely identify data items.

2. special formatting used to separate and highlight display tables, inserted as "constant" data items.

3. informational data of general interest.

Primary keys are always put into a display; the other two kinds of mandatory fields are included depending on how specific a query is about the data items to be retrieved by a query. All mandatory fields and the level of specificity at which they apply have to be identified in the mandatory fields table for a given application.

3.5.2 Display Headings

To help a user read AQF displays, two types of headings are produced. The first consists of column headings for retrieved data items taken directly from field specifications in an intermediate query. The second consists of labeling columns accord to units of measurement for numerical data; this units information is stored in each of the tables where target data items are defined: the field name correspondence table, the sub-key table, and the mandatory fields table.

3.5.3 Virtual Fields

Target data encoded in exotic ways may not be immediately displayable; for example, a bit encoding of color. To prepare this information for display, it is necessary to convert such encoded data first into a string or other more readily interpretable format. AQF accomplishes this by allowing encoded data to be designated as virtual fields, marked by having negative offsets in a target data record type. These virtual fields are computed as on the fly through a special AQF entry point (COMPUT) to call generation procedures defined for given target data bases.

Virtual fields included in the mandatory fields table can be used as a device for report generation. The procedure for computing a virtual field can be employed to produce arbitrary output based on the contents of a target data record instance. Various kinds of output formatting can be obtained in this way, including the delineation and highlighting of data items.

3.5.4 Arithmetic

AQF report generation in Pass 4 can be set up to compute various functions on a numerical data field and to display the results. The particular functions executable for a given AQF application are defined through special AQF entry points (DETFNC, COLFNC, and PRTFNC). The AQF demonstration systems currently can compute sum, total, minimum, and maximum; other similar functions can be defined as well. The problem of more general arithmetic capabilities on several fields at a time is discussed in Section 6.2.2.

3.5.5 Meta-Queries

When a query makes no reference to any field in a relational hierarchy, various different responses are possible. One possibility is to simply print a diagnostic message and to disregard the query, but this is rather obtuse, given that the query is intelligible. A second possibility is to dump key portions of all data accessible to the user, but this is probably not a good idea for data bases of any significant size. A third possibility, which is implemented in AQF, is to interpret the query as a meta-query about the structure of a data base rather than about its content; this is a convenient way of providing online data base documentation.

AQF stores prepared text describing each of the relations in a relational hierarchy logical model. The description of a relation is displayed when an intermediate query marks that relation with a "(?)" but makes no reference to any field. The text descriptions are entered at the time that a relational

hierarchy is set up as a logical data model. AQF provides a special input program to convert text from an ASCII input file into the proper form for retrieval and display.

3.6 SORTING

AQF implements sorting only on fields of the output display produced by Pass 4. This is done in Pass 5 of AQF, which serves as the output module for AQF; Pass 5 calls an AQF sort subroutine employing a standard partition-exchange sort algorithm. The incorporation of sorting in a separate pass lets AQF sort an output display entirely within an internal buffer. It is assumed that such displays will never be much larger than the size of a typical CRT screen of a user terminal.

Sort specifications are compiled automatically by AQF from a user's query. Sorts may be in ascending or descending order; output fields are sorted either alphabetically or numerically depending on their original data type; and two levels of sort field priority are defined, allowing for simple grouping of output data. Sorting can be specified separately for the individual segments of output produced for each data access sequence derived for a query.

4. COMPARISONS

AQF provides an extensive range of capabilities for data base access and in particular is useful for correlation of data from different data bases. Its actual value as a part of an overall interactive information system depends on a variety of factors:

- o The degree of training expected of users.
- o The predictability of information requests by users.
- o The complexity and size of target data structures.
- o The type of computer hardware available.

How these factors affect the applicability of AQF in a given system is best seen by looking at various alternatives to AQF and weighing the relative advantages and disadvantages. This section will look at three main categories of interactive systems for comparison: menu-driver query systems; formal language systems, including most relational data base systems; and natural language systems of various types.

4.1 MENU-DRIVEN SYSTEMS

In a menu-driven system, the user selects data for display from various fixed options. Typically this is set up with several levels of options, where selection made at higher levels determine the availability of selections at lower levels. The scheme is straightforward to implement; and it is easy to use, especially in conjunction with graphic input aids like the light pen. It is probably the best approach to take when a data base has a fairly simple structure and when queries are predictable.

The menu-driven system, however, tends to be inflexible. Major changes to a data base or to the repertory of allowable queries and associated responses all require reprogramming of the system. This approach does not lend itself to applications where information needs are evolving or where access to data is on an exploratory basis.

The entering of a query through a menu can also be inconvenient at times. Although the number of manual operations to enter a query is reduced with menus, the user is often forced to look through a great deal of irrelevant data in order to make a selection. The actual selection process itself can be highly unnatural if a user has to repeat a series of selections many times for sequence of queries different only in a single detail.

Another difficulty with menu-driven systems is at the output end. Such systems tend to have only a few ways of displaying information, and this may consist of showing the contents of an entire target data record even though most of that data is of no interest to the user. With fixed displays, there is typically no easy way of correlating and comparing values across data records.

In general, classical menu-driven systems are most useful when data is to be processed on a production line basis or when the number of menu options is small. In any kind of analytical situation where the structure of data is complex too, a user needs much more flexibility in looking at data, and the support capabilities of something like AQF become quite attractive. Relying solely on menus also becomes impractical as a data base grows to the size where there are too many retrieval keys to list in menu displays.

One interesting possibility here is to combine AQF with a menu-driven approach. Instead of a natural language interface as implemented in Pass 0 of AQF, one can substitute a menu-driven front-end with provisions for manually entering retrieval keys too numerous to list. This would combine the data base modeling, data correlation, and report generation capabilities of AQF

with the simplicity of a menu interface. It should perhaps be noted that menus could be into much more powerful query entry tool with two dimension displays and color graphics.

4.2 FORMAL LANGUAGE SYSTEMS

Where flexible access to online data is needed, formal query language systems are usually implemented. These allow users to express information requests in a highly logical language that is well-defined syntactically and semantically. The most prominent example of these are the retrieval languages designed for commercial data base management systems and the various formal user interfaces designed for relational data base systems (c.f. [1][2]).

All formal query language systems require that the user learn an artificial language, although in some case it may masquerade as being natural by having words in place of mathematical or logical notation. This extra demand on the user is usually justified on two main grounds: first, that a well designed artificial language is much easier to process by computer than a natural language; and second, that the artificial language would be more precise. If neither were so, then there would hardly be any need of an artificial language at all.

There are, however, problems with formal query languages in that they closely resemble programming languages in their usage. Accordingly, formal query languages are most suitable for persons who can readily learn a programming language, meaning that most people will not take to a formal query language quickly. This difficulty is aggravated by the fact that formal query languages tend to be arbitrary in definition anyway and will often be somewhat inconsistent from one system to the next.

Implementation problems also arise with formal query languages. Almost all such languages are predicated on relatively simple logical models of data that seldom correspond to the complexity of actual target data bases. Full

use of these languages requires that existing data be reformatted to correspond to how their logical data models look; for example, a user might have to convert an entire data base into a relational representation. This makes a query language much less useful than it might be.

The development of AQF addresses most of the issues raised here. The AQF language is very natural and easy for non-programmers to learn, but yet it can be processed readily enough even on medium-scale hardware. The biggest advantage of AQF, though, is with large existing data bases of complex structure because AQF can work with such data without any prior reformatting. This makes it possible to develop a query capability for a target data base without disrupting any data processing applications already supported by it.

4.3 NATURAL LANGUAGE SYSTEMS

There are two types of natural language systems that need to be considered here: those that focus on natural language as a means of exhibiting intelligent machine behavior, and those that look at natural language usage as a source of ideas on how to improve communication between computers and users. AQF is of the latter type.

4.3.1 Machine Intelligence

Intelligent natural language systems, of which SHRDLU [9] is probably the best known, seek to understand language in the ways that human beings seem to understand it. This encompasses such problems as recognizing all the implications of a given sentence in a given context, filling in details that are expected to be understood, and devising effective procedures for dealing with pathological examples of language. These problems almost always have to be approached through the compilation of large bodies of online world knowledge and elaborate inference schemes.

Such technology is as yet not mature enough to build practical software with. More significantly, however, it does not really appear necessary when the goal is only to be able to request certain that items of data be retrieved from a data base. So instead of aiming for intelligent behavior, AQF seeks to develop simple, reliable tools to support interactive access to data bases at reasonable cost.

For such reasons, AQF currently supports no general inference capability and deals with no world knowledge other than the information in target data bases or in a logical data model. These remain possibilities for the future. Inferential techniques will be practical for query access facilities when they can be made fast enough for interactive operation; employment of extensive world knowledge to support general query access will be feasible when there is a systematic way of constructing world models applicable to particular target data bases.

4.3.2 Technology Transfer Systems

In the past few years, a practical approach to building natural language systems has evolved. The premises of this approach are that a large body of proven technology exists for natural language processing and that much of the technology can be directly applied to improve the capabilities of software for applications like interactive data base access. Work along these lines has been promising (c.f. Section 1.2.2), leading to scores of efforts to develop practical natural language systems of all kinds.

In the area of data base access, most systems including AQF take the view that it does not really take much more trouble to go from a formal query language to a reasonably natural query language. The problems of parsing and interpreting natural language are fairly well understood, and if solutions do not yet exist for all of them, they can at least be worked around in the special case of data base access. Any extra overhead involved in processing natural language queries in any event turns out to be relatively insignificant

compared to the normal overhead of searching for and retrieving items from a target data base.

Natural language access based entirely on extension of formal query language processing capabilities, however, inherits the problem of applicability to existing target data bases. Their usefulness is diminished when they require data to be converted into a special format like relational data structures. Where target data is not already in a convenient format for natural language access, the approach of AQF is helpful because its data base mapping capability eliminates the need for any conversion.

The point to note again here is that AQF is not simply a natural language query processor; the natural language front end of AQF can easily be replaced by something altogether different. The particular virtue of AQF is that natural language is well-integrated with versatile data base access capabilities. This makes AQF most useful in situations where data base access is actually a serious problem instead of merely being a little inconvenient.

5. GETTING UP

AQF is available for distribution in the form of FORTRAN source files. These can either be incorporated into an existing data base interface or employed as the nucleus of a separate query system. This section will outline the basic procedures involved in setting up AQF for use from its distribution source; a more complete description with examples is given in the AQF User's Manual.

The basic steps to AQF setup are as follows:

- o Define a logical model for target data of interest. This is a relational hierarchy.
- o Define target data record linkages pertinent to the logical model. This may involve some programming.
- o Compile a query language vocabulary for the logical model.
- o Implement any special target data access methods.
- o Implement any special data type conversions.
- o Define mandatory fields for report generation.
- o Compile and link AQF from supplied command files.

Most of the work here involves setting up various tables for AQF. In AQF, all tables are produced from source text files by special support programs, one for each type of table. the AQF query language grammar and an AQF dictionary are treated as tables.

Any necessary programming of AQF for a target data base is restricted to predefined entry points in specific AQF modules.

5.1 DATA MODELING

The first priority is for a system manager to determine what data AQF should work with. This is done by generating a relational hierarchy as a logical model of target data bases for AQF. The model would include all relations defined for the AQF application, their hierarchical ordering, and all the fields defined for them. Fields of relations need not correspond to target data items, but should be at least directly computable from data items. Not all data items need to be included in a logical model.

5.1.1 Relational Hierarchy

The definition of a logical model for AQF departs from standard procedures of data base design in that the names of relations must come before the relations themselves. In effect, the choice of relations and their ordering in a hierarchy are determined by the prior selection of the relation names for the hierarchy. This selection is on linguistic grounds; a word is made a relation name if it occurs fairly often as a modifier in natural language designations of target data items. The ordering of relation name words in the designations then defines the hierarchical ordering of relations, and fields are inserted into this hierarchy at the lowest relation name in a downward chain of relation names modifying the field.

There is no hard rule on whether to make a given data designation word into a relation name or into part of a field name. This depends on the applications for AQF. Usually, it will be advantageous to have as many relations as possible because the greater articulation of a logical model means more possibilities in report generation. This, however, has to be balanced against the fact that more relations also means longer target data access paths and more target data record instance lists to maintain.

The relation names for a hierarchy go into the AQF relation name table; this also specifies the immediate ancestor relation name for each entry. Descriptions of each relation go into a separate AQF relation documentation table, which supplies the text to be displayed when a meta-query about the structure of a data base is submitted by a user.

5.1.2 Linkages

Once a relational hierarchy is defined as a logical model for AQF, it has to be mapped into target data bases. The first step here is to define a field name correspondence table showing the location of data items for fields in terms of the target data record types containing them, their data type, and their position in a record. If there are any virtual fields not corresponding to any data item, then these should be noted in the table with a negative offset position for some record type, and code to compute this field should be inserted in the AQF virtual fields subroutine.

With the field correspondences, each field can be identified with data items at a point in access space. The next step is to define the access linkages that will tie these items together. The goal is to have a set of links such that for each data item, there is a data access sequence composed of links that satisfy two criteria:

- The relation coordinates of an access sequence for a data item correspond to moving from a top level of a relational hierarchy down to the relation containing the field designating the item.

- The record type coordinates correspond to a chain of accesses starting from a directly accessible record type (e.g. by sequential or indexed access) down to the record type for the data item.

Links crossing between relations are defined in the inter-relation link table:

links staying within the same relation, in the intra-relation link table. Code defining the linkage mechanisms should go into the AQF linkage procedures (FIRSTR, MLTJPR). In addition, all indexed fields should be identified in the AQF table for them and the indexing methods added to the AQF indexed-access subroutine (INDXMP).

5. VOCABULARY

The basic AQF query language grammar defines the syntactic function words like "a," "the," and "is," which make up the skeleton of natural language phrases. All the content words of a query language, which refer to a particular target data base, must be defined in a dictionary table for a particular target data base. The selection of content words will depend greatly on the expected users of an AQF system.

The general rule for vocabulary is to include all words that can be interpreted as being a relation name, part of a field name, or an explicit literal value associated with some field. These are entered into an AQF dictionary by assigning them to one of just over a dozen possible parts of speech and establishing their relational hierarchy referents. Where there is possible ambiguity over referents, AQF allows for definition of a word with a special part of speech, which is processed so that Pass 2 will take on the responsiblity of establishing the referent.

5.3 SPECIAL ACCESS PROCEDURES

For modularity, all AQF target data base access is through a single subroutine (ACCESS), which accepts a record type and record number as input arguments to indicate a particular target data record instance and returns that record instance in a given buffer. This procedure cannot be supplied with a standard AQF distribution package because it must be designed for the target data base and its data base management system. Writing the record access subroutine can be simple or complex depending on whether the

organization of the target data base has structures corresponding closely with the nominal record types recognized by AQF. Some remapping of data items may be necessary.

In addition to the record access subroutine itself, the system manager setting up AQF is responsible for supplying two special subroutines (INACCS, DEACCS) serving to attach AQF to a target data base and to detach it. Again, these subroutines may be trivial or complex depending on the target data base. A set of access procedures designed for data bases consisting of FORTRAN sequential-access and direct-access files is available with AQF.

5.4 TARGET DATA TYPES

AQF currently recognizes only three data types: integers, floating point numbers, and character strings, with integers and floating point numbers being either single- or double-precision. Other data types in a target data base must be converted using AQF virtual field definitions and conversion code called through a special AQF entry point (COMPUT). This is the responsibility of the system manager setting up an AQF system.

Users may deal with data only of the types recognized by AQF, whether this be actual or virtual data. Addition of other data types is possible, but would involve significant changes of code in AQF passes 3, 4, and 5. This is because the detailed characteristics of a data type must be taken into account when matching target items, formatting them for display, and sorting on them.

5.5 MANDATORY FIELDS

The notion of mandatory fields is the basis for AQF report generation. The system manager setting up AQF must establish the various key fields, labels, and default output fields that are to go into AQF output to make it more readable and informative. This is accomplished by putting mandatory fields at appropriate points along an access sequence so that they will be

inserted into the relational n-tuples composed by AQF in response to a query.

The coordination of access links with mandatory fields allows for the tailoring of output displays for users. The choice of access links governs the kinds of relational n-tuples that can be produced as output; the choice of mandatory fields governs the type of information appearing in a relational n-tuple, exclusive of that specifically requested by a query. The access sequence for requested data item thus establishes an associated output format.

A system manager has flexibility in defining output formats because it is possible to define more than one distinct access sequence following the same target data base record linkages. This can be done by defining unnamed invisible relations in an access space that map into the same part of a target data base as a named relation. The existence of such parallel relations allow for the generation of data access sequences that are completely equivalent except for the mandatory fields associated with them.

6. FURTHER WORK

The current AQF software package offers a broad range of services to the non-expert computer requiring interactive access to online data. AQF as it stands now, however, has much potential for continued evolution. The basic multi-pass framework of the original AQF demonstration system has proved to be extremely workable, lending itself readily to the incorporation of many new ideas.

This section will examine AQF not as a finished product but as a concept of much wider scope. Various improvements and extensions of AQF will be considered, all being practical undertakings. The important question to be raised here is really not whether something can be done, but whether it ought to be done in light of user information needs.

6.1 IMPROVEMENTS

There are a number of straightforward ways to enhance present capabilities without having to develop any major new algorithms. These would be areas of continuing work.

6.1.1 Grammar Improvement

The current AQF query language grammar consists of about 500 rules and definitions describing a small subset of English. It has been developed extensively through experimentation starting with the original AQF effort, but more work remains to be done to extend the syntax handled by it. Rule storage space in the current implementation of AQF could easily handle another 200 rules, so that there should be no problem with room.

The AQF parser plus grammar needs to be exercised by many more experimental users since the acquisition of any kind of language inevitably

must come through exposure to many different examples. Persons unfamiliar with AQF are very helpful in coming up with valid queries that are rejected by AQF. In most cases, these point out areas where the basic AQF query language grammar could stand improvement.

6.1.2 Code Optimization

Because AQF is experimental, its implementation was through a robust programming approach that reduced the probability of errors, but probably resulted in inefficient code. For example, a serious problem in this respect for the most recent AQF demonstration system is with target data record buffers; at present, only one buffer is available, forcing much rereading of data during data searching involving several data record types. A better buffer area management procedure would probably be helpful here.

Other aspects of AQF where optimization would be appropriate are in access path generation, data conversion and packing, and output formatting. Data base search and retrieval might also be open to improvement, but this is probably something that cannot be improved upon within AQF alone. Output formatting is a good possibility for optimization simply because it is so large now.

6.1.3 Error Reporting

The diagnostic output produced by AQF in response to an uninterpretable query is currently at a primitive level. The AQF query processor for example could do more to indicate why a query could not be parsed; an error message might show how far an analysis got and what unknown words appeared. Similar improvements in error messages could be made in the other AQF passes for various overflow conditions.

6.2 EXTENSIONS

Although the current AQF demonstration running under VMS compatibility mode is already fairly large, there is still room for the addition of a few major modules. Which additions to incorporate, however, has to be established according to need.

6.2.1 Spelling Correction

Because spelling or typographical errors tend to be frequent in keyboard entry of text, most natural language systems incorporate some capability for their correction. This usually is called upon to process an unrecognizable word, with the typical procedure being to look the word up a table to find what might have been meant and then to present these possibilities for the user to respond to. Some research on how to do this has been carried out under the AQF effort, but nothing yet along these lines has been integrated into the AQF package.

For AQF spelling and typographical correction, the approach would be to implement a fairly simple scheme to catch some of the most common problems; letter transposition and wrong choice vowels, for example. AQF would maintain an external file of target data reference words and English syntactic function words that are likely to be misspelled. This would be applied both for automatic correction and for corrections where the user has to choose between possibilities.

A possible scheme for AQF would be to index words in the misspellings file by their consonant occurrences irrespective of ordering in a word. Candidate corrections obtained from this indexing for a misspelling could then be filtered further according to length, vowel occurrences, and other measures of similarity with the misspelling. This sort of procedure would be incorporated in the lexical analysis part of the AQF pass 0.

6.2.2 General Arithmetic

Systems like REL [7] allow queries to specify computations on different data fields of a logical model. Such a capability may be useful for AQF; it could be implemented as extension of the arithmetic function computation already in AQF. AQF would maintain special registers for carrying out these operations and allow these registers to be referenced in queries as virtual data base fields. More elaborate capabilities than this would probably be unnecessary in AQF, given that AQF is not intended as a kind of interactive programming language.

6.2.3 Arrays

The current AQF scheme for handling arrays of data elements is awkward for larger arrays and for multi-dimensional arrays. This could be remedied by building into AQF a specific array bounds handling mechanism, which would allow references not only to one array element at a time but also to an entire range of array elements. Unformatted text data fields could also be handled better with this capability.

The main changes to AQF would be in the way that matching target data record instances are maintained and in the report generation and output procedures. AQF record linkage procedures (FIRSTR, MLTPSR) would be employed much the same way as before. The overall scheme would remain fairly simple, although this would probably provide more array handling capabilities than found in any other natural language data base access facilities.

6.2.4 Fuzzy Matching

Internal arithmetic operations on target data in AQF now is entirely with fixed-point numbers, and matching for equality of values must be exact. This, however, will probably be inconvenient or even unacceptable in many data access applications and especially so where unit conversions or other

computations introduce round-off errors. It would be helpful for AQF to allow for fuzzy matching of numerical values within some range of tolerance.

The easiest approach to fuzzy matching in AQF would be to extend its notion of data type to include a range of tolerance expressed as a percentage. This could be implemented with additional code in AQF Pass 3 and with a few minor changes in AQF data structures and mapping tables. This could be done without having to do anything else with the rest of AQF.

6.2.5 Negation

AQF currently does not permit negation in queries, but this could be added in a fairly straightforward way. There are only two possibilities where negation might be applied to intermediate queries: on numerical values in query markers of the form "(n!)" and on comparison operators relating a value to a field. The first case would be best handled by encoding query counts differently to include a comparison specification also; the second would require only minor changes in Pass 3 field matching. Most of the work involved in handling negation would be in extending the basic AQF query language grammar.

6.2.6 Macro Expansion

Interest has been expressed in having AQF support a "macro" string expansion capability comparable to that in REL, where the user can specify that occurrences of a given "macro" string in a query be interpreted as an expanded definition string. The definition string usually would include dummy parameters that would be replaced by matching strings associated with the occurrences of the "macro" string in a query. In this way, users could develop a personal form of shorthand for queries.

In AQF, such a capability would be difficult to duplicate because the AQF parser by itself is not as powerful as the REL parser in input string manipulation. A possibility for AQF is to define a macro expansion in terms of dependent clauses to be inserted into an intermediate query during the rewriting phase of Pass 0. This, however, would require extensive work, and it is unclear how much use it would be.

7. CONCLUSION

AQF at present is in a situation like the early days of compilers. The existence of AQF at all shows what is actually possible to accomplish with available resources and points in a direction for future work. It should ultimately be possible to build a much better data base query facility than the current AQF implementation, but AQF in the meantime will have helped to clear the way by sharpening issues pertinent to developing more powerful query facilities.

The AQF software package is particularly workable for both applications and development because of its modularity. It can readily be made to fit on processors with limited address space, and it is easy to change. The basic software has been run by a variety of users since the initial operation of the AQT demonstration system two years ago. The FORTRAN source for this software is available for experimental use.

7.1 CAPABILITIES

The design of AQF aims at usability through simplicity. Although AQF breaks no new ground in terms of machine intelligence, it manages to provide a full range of important capabilities for support of data base access.

7.1.1 Natural Language

Natural language in AQF is a method to make access to a data base as transparent as possible. It is not supposed to eliminate user training entirely; instead, it allows a user to learn a query language that is analogous to what the user is already familiar with. This makes it easier to describe the kinds of restrictions for communication with AQF. In real life, people seem to know already how to make allowances in talking with persons lacking a full grasp of language, if such shortcomings themselves are natural

in some sense.

In contrast to most natural language systems, AQF query processing avoids the problem of trying to recognize sentences. It focuses more on applying the various conventions about linguistic usage that allows someone to designate an item of data and its relationship to other data. This permits query processing to be more simple and flexible in dealing with imput. Syntactic analysis is fast enough to be negligible in an information request, almost always in about a second at a user's terminal on a time-shared system.

7.1.2 Multiple Data Bases

The main advantage of AQF is its applicability to existing data. It adapts itself to target data bases, rather than forcing data to take certain formats or to be organized under specific data base management systems. AQF is particularly useful when target data is distributed over several different data bases, each with dissimilar user interfaces. The natural language capability of AQF here would provide a convenient common data access language.

More significantly, however, AQF offers the capability of generating information displays correlating data from different data bases. This supports more effective analysis of online data as well as improving overall access to existing data bases by analysts who may be non-expert computer users. No other natural language systems implements such a range of services.

7.1.3 Report Generation

This is perhaps an underrated aspect of information system design. Although there is great concern about making information requests more flexible and easier for users, there has been no comparable effort for the flexible display of data in ways easy to interpret by users. In many cases, there is a lack of distinction between data as opposed to information needed by a user.

The approach in AQF is to avoid the mere dumping of data values requested by a user. The semantic dependence of data items is also taken into account in order to produce coherent output for a specific query. Unlike most interactive information systems, AQF output is not restricted to a dozen or so standard formats.

7.2 USER DEVELOPMENT

The effectiveness of the AQF approach can be assessed only by applying AQF to actual data bases and letting it serve real users. In this way, one can see in practice whether capabilities like natural language, relational data models, table-driven data base access, and automatic report generation can help anyone out significantly. If AQF does prove to be viable here, then it can be developed further within a context of real information needs.

A perennial problem with truly new information systems in that they seldom fit into the information flow patterns of organizations accustomed to working with greatly limited information processing. The importance of a new system may in fact ultimately be to alter information flow patterns radically to improve the overall capabilities of an organization; but this cannot be accomplished at a single stroke. A system must grow gradually into an organization so that the organization can develop the necessary procedures to take full advantage of the system.

The adaptability of AQF works out well in this respect. It permits new capabilities to be introduced in a fairly inobtrusive way into the information flow of an organization. There is no need to convert existing data bases or to acquire special support software or special hardware. It is possible to try AQF out within an organization without incurring heavy costs.

One interesting possibility here is to implement AQF on a micro-processor. For example, there are commercially available CRT terminals that can accommodate a 16-bit DEC LSI-11/03 micro-processor, up to 128K bytes of

static MOS memory, serial interfaces, and a dual floppy-disk drive at a total price under $12,000. Because RSX-11M can run on such a micro-processor system, the present FORTRAN version of AQF could immediately be implemented on the system, although a version with tighter assembly language code might be more desirable.

The idea is to put all of AQF except for target data base record access on a terminal that could be attached to any computer system supporting current-loop terminal interfaces. To bring AQF up, it would only be necessary write a target data base access program to run on the host processor and to communicate with AQF on a satellite terminal processor. The entire operation of AQF here could be made completely invisible to the host operating system. All AQF tables would be maintained at a terminal on a floppy disk, with the user able to change a logical model by simply inserting another disk.

7.3 EVALUATION

At the start of the AQF effort, it was intended that an AQF demonstration system be set up with real data for trial operation by volunteer users, analysts who would ordinarily work with the data. This was to provide the basis of an operational test of AQF; but because of a variety of reasons beyond the control of PAR Corporation, neither data nor users were ever identified, making an operational test impossible. The problem may have stemmed in part from a curious paradox afflicting natural language systems: although much lip service is paid to importance of natural language communication with computers, it is hard to think of practical situations where it might actually be applied to advantage.

To show the capabilities of AQF despite lack of real user data, it was necessary to bring up two different AQF demonstration systems with two different locally generated data bases. These illustrated what was possible with AQF: in one case, running independent of any data base software; and in the other, interfacing with a basic data record management facility (RMS-11).

The AQF software developed for the latter case would also apply to data bases generated with the SABRES data base management system, which is built on RMS-11.

The two demonstration systems generated from identical code for the six passes of AQF show that AQF is insensitive to the organization of target data bases. Both systems run quite fast in PDP-11 compatibility mode on a VAX-11/780. Both have been tried during demonstration sessions by various users with only minor difficulties; in almost all cases, shortcomings identified by users could be remedied shortly afterward with changes in the basic AQF grammar or in the six passes of AQF.

7.4 PROSPECTS

Although AQF is not absolutely portable to all computers, its being written in FORTRAN makes it reasonable to move to more different types of processors than any other natural language system. As online data bases become more widespread, AQF techniques with or without natural language should provide a viable option of increasing importance for implementation of interactive data base access. The only major requirement now is the willingness of users to experiment with new techniques like that of AQF.

REFERENCES

[1] M.M. Astrahan and D.D. Chamberlin. Implementation of a structured English query language. Comm. ACM 18 (October, 1975), pp. 580-588.

[2] R.F. Boyce, D.D. Chamberlin, and W.F. King. Specifying queries as relational expressions : the SQUARE data sublanguage. Comm. ACM 18 (November, 1975), pp. 621-627.

[3] E. Codd. A relational model of data for large shared data banks. Comm. ACM 13 (June, 1970), pp. 377-387.

[4] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complete data. ACM Transactions on Database Systems 3 (June, 1978), pp. 105-147.

[5] C. Mah and J. Morris. Advanced Query Techniques for S&T Intelligence. RADC-TR-620, Rome Air Development Center, October, 1979.

[6] V. Pratt. A linguistics oriented programming language. A.I. Lab Memo. No. 277, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, February, 1973.

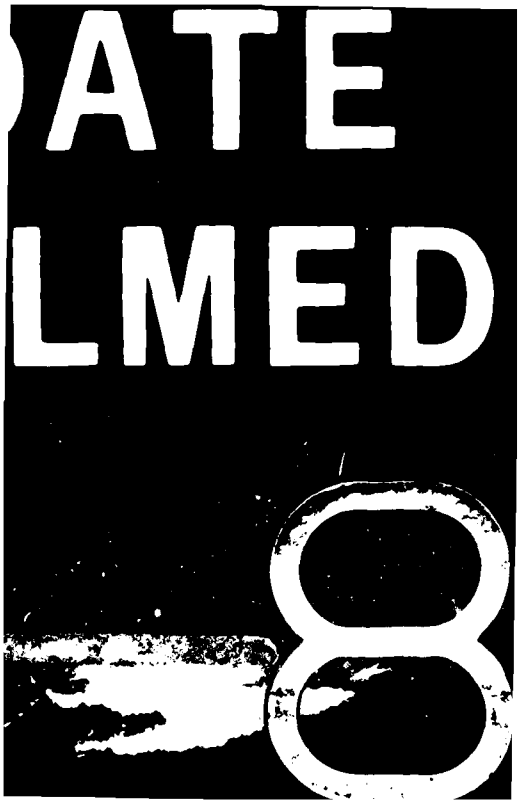
[7] F. Thompson and B. Thompson. Practical natural language processing: the REL system as prototype. In Advances in Computers 13, M. Rubinoff and M.C. Yovits, eds., New York: Academic Press, 1975.

[8] D. Waltz. An English language question answering system for a large relational data base. CACM 21 (July, 1978), pp. 526-539.

[9] T. Winograd. Understanding Natural Language. New York: Academic Press, 1972.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


ATE
LMED
8